

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

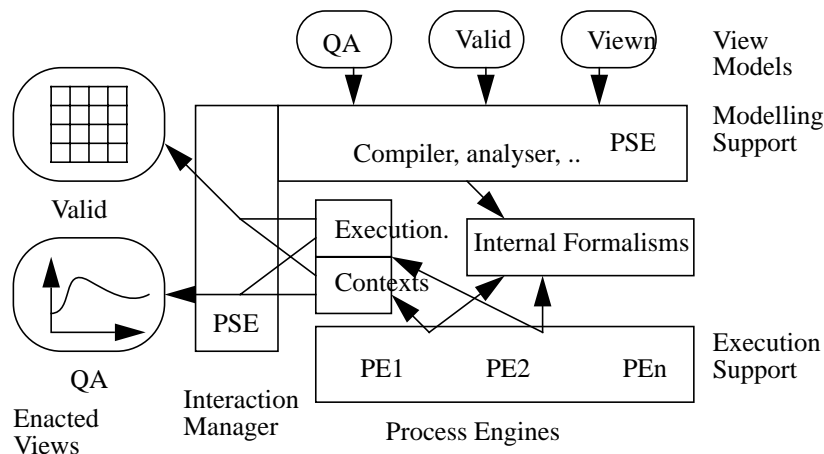
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

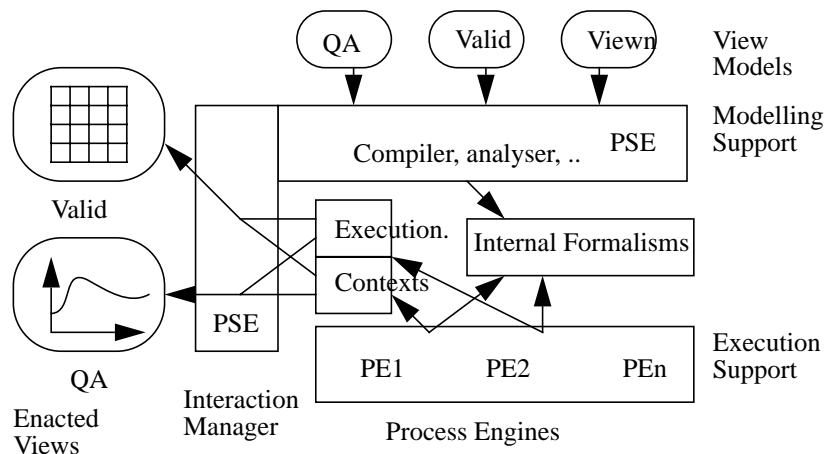
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at <ftp:imag.fr>, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

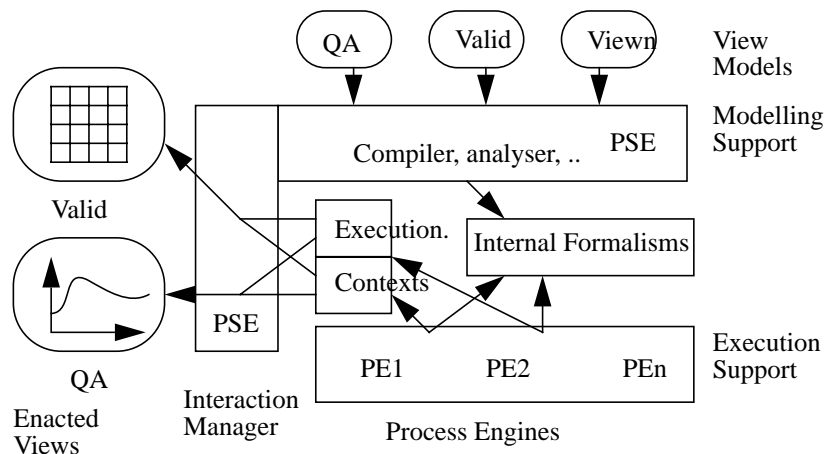
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Wokshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available bt ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

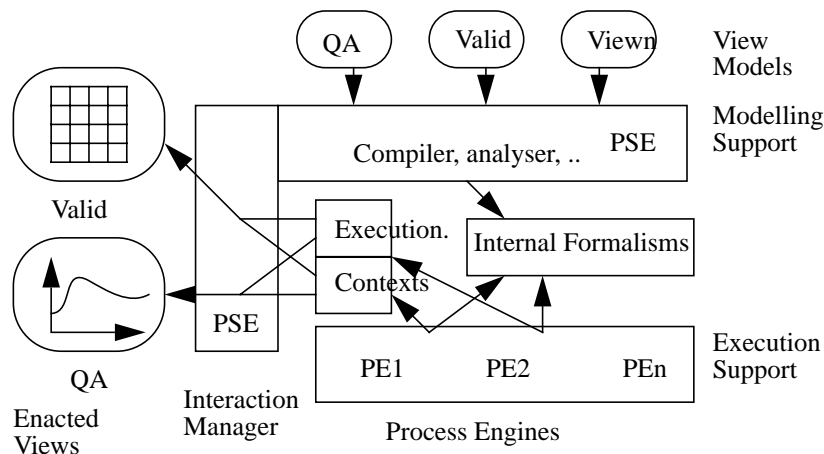
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut für Informatik, Universität Zürich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

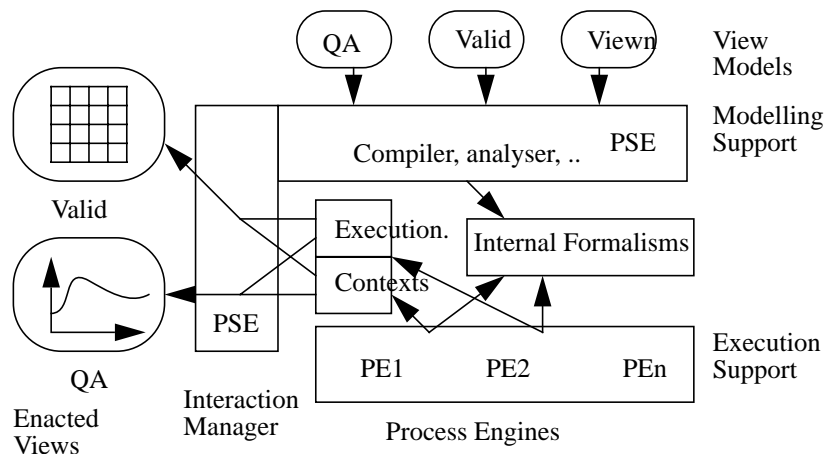
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

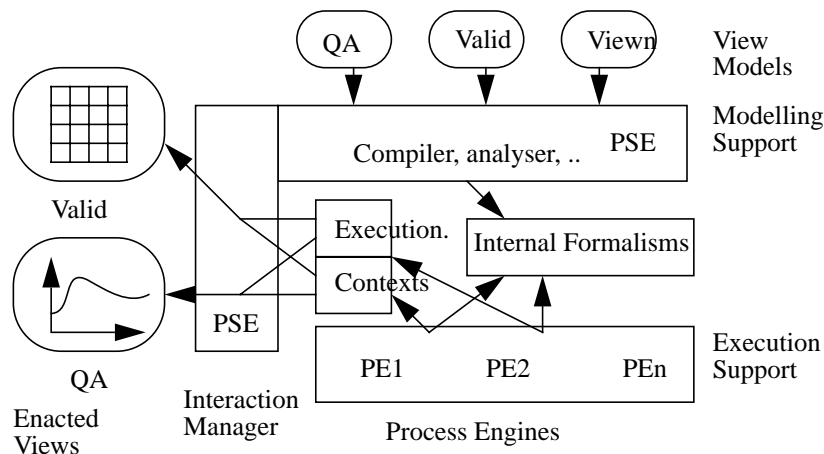
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at <ftp:imag.fr>, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides a uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

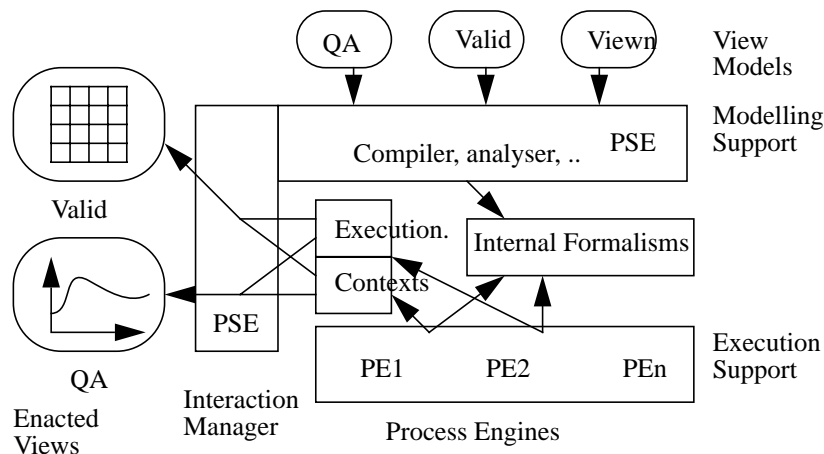
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

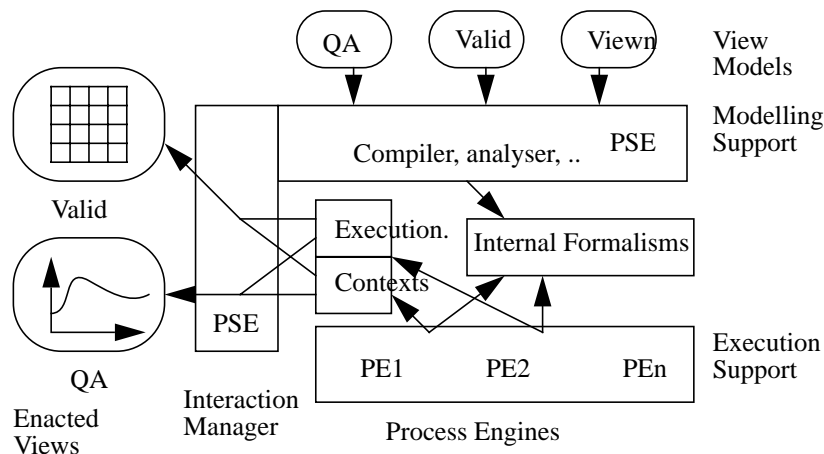
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at <ftp:imag.fr>, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides a uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

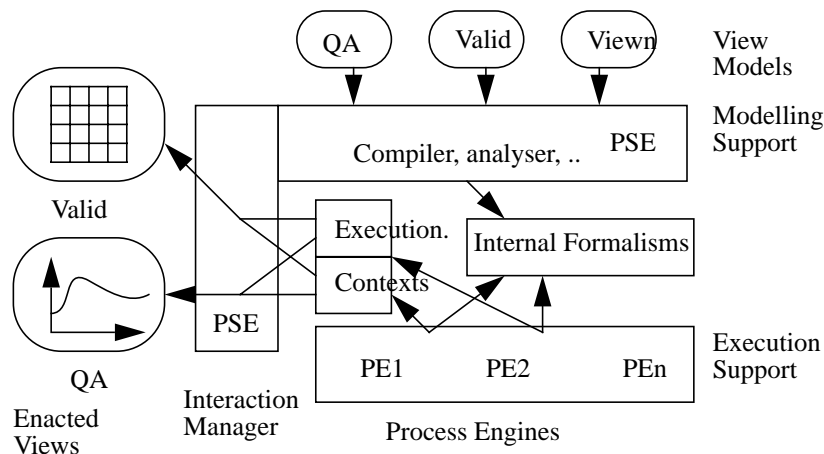
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at <ftp:imag.fr>, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

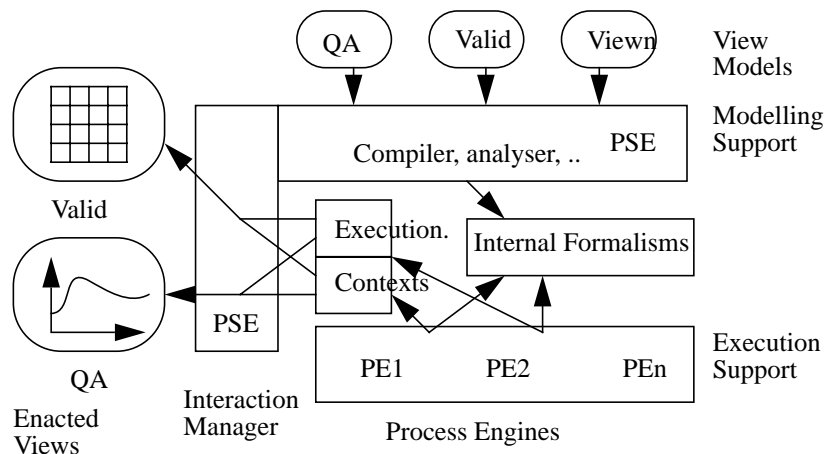
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut für Informatik, Universität Zürich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

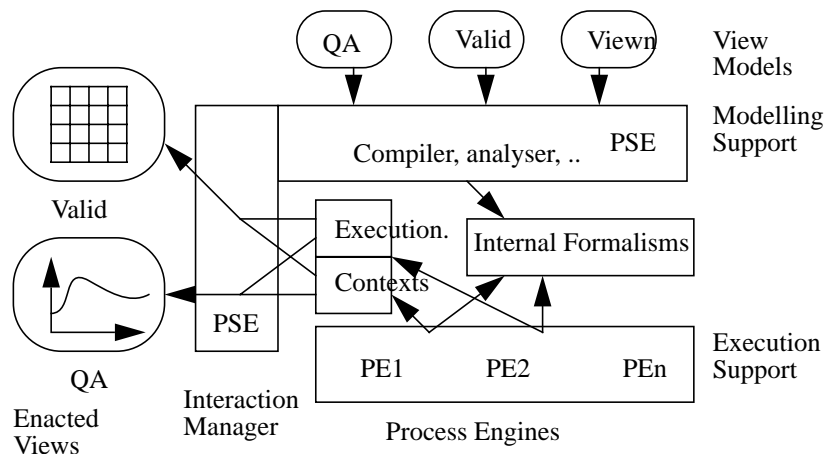
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at <ftp:imag.fr>, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut für Informatik, Universität Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides a uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

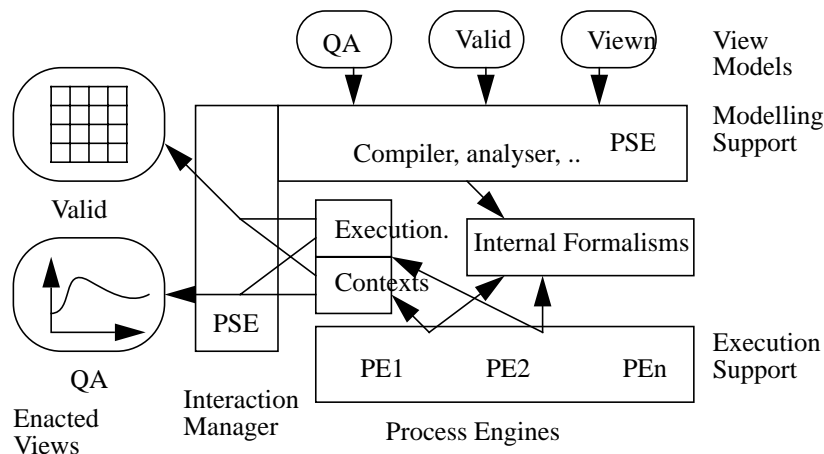
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Wokshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available bt ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

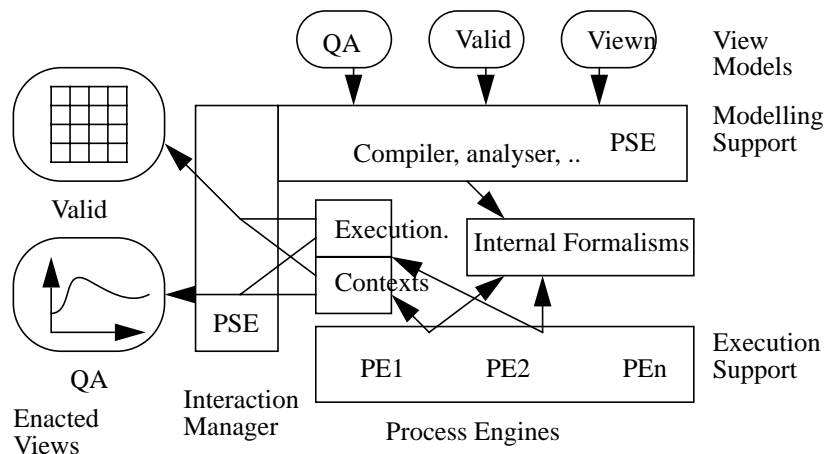
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

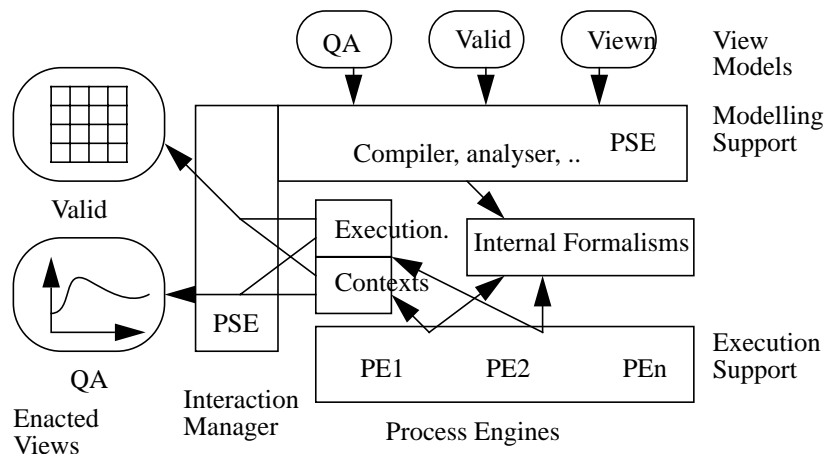
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at ftp.imag.fr, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut für Informatik, Universität Zürich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

A Generalized Multi-View Approach

Jacky Estublier and Nouredine Belkhatir
L.G.I. BP 53X 38041 Grenoble
FRANCE

Abstract. It is advocated here that integrating abstraction and modularity into the concept of point of view, and extending the view concept to the process itself (and not only to data used by processes), provides an uniform conceptual framework for aspects like agent point of view, quality models and process monitoring..

1 Introduction

Within Process Modelling, formalisms have been proposed to satisfy requirements like modularity and , abstraction. On the other hand, formalism have been proposed to provide multi-views [1, 3, 4] but they do no meet the previous requirements.

We propose to extend the concept of view point in a conceptual framework integrating modularity and abstraction to structure the process. Since “Software processes are software too” [2], it is possible to reuse technology and concepts borrowed from languages (modularity, encapsulation) and Software Engineering (configuration, abstraction), still retaining the point of view approach.

A software process is in fact the aggregation of numerous process fragments; each fragment describes different part of the overall process, with no overlapping. It is the usual approach, but it is far to be the reality.

In any large organization, each actor, or class of actors, has a *partial and overlapping* view of the complete process. This view corresponds to the process part this agent needs to be aware of, and should be expressed in the terms this agent is familiar with (its universe of discourse).

Most processes are multi-agent processes, i.e. *multiple agents* are collaborating in a single process, as it is the case in meetings, but also in most activities, like change request (involving both team leaders, designers, developers, validators and so on). It is a common oversimplification to consider activities as being independent and undertaken by a single agent.

A view can also be an *abstraction* of the “real” enacted process; it “sees” a limited part of the complete process, and represents a common reality under a specific presentation. Quality and monitoring are often abstract views.

The concept of view has been involved in different computer technologies (process, design, DB, AI, SI and so on) but under many different definitions. In our work, assuming a process model exists, (called the reference process), a view is defined as a process definition where parts of the reference process are missing (to be ignored in that view), parts are renamed (to fit the concepts known under the view), and parts are abstracted (reducing the level of detail to what is relevant in that view). “Part” here refers both to product and activity aspects.

In our work,

- Views can *share process instances*,
- A view is *enactable* either to monitor existing processes (observation process) or to interact with existing processes (actor process).
- A view can be defined a *posteriori* on an existing enacting process.

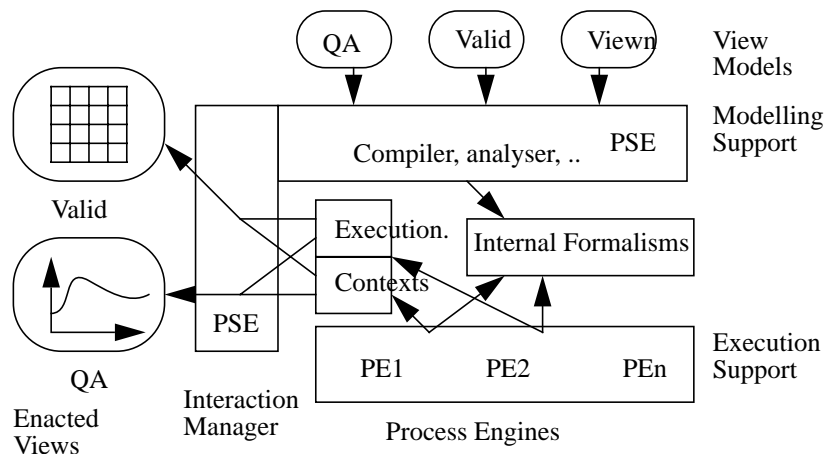
No multi-view formalism meet these requirements. In Tempo [1], views are applied in the product only, in [3] views are not enactable, in MVPL [4], they do not share process instances.

Since views can overlap, there is a risk that different views may define the same activity inconsistently. Since views are enactable, the same activity can be carried out twice. It is not easy to ensure consistency between the different views [3, 4]. The following requirements are particularly important:

- Process overlap should be detected.
- It should not be possible to duplicate activities,
- It should not be possible to define the same process in an inconsistent way.

An example of our approach is the following. Let there be a validation process (valid), and a quality assurance (QA) process. The former executes the technical validation activities; the later sees a part of these technical activities and also records, for measure and traceability purposes, some selected events, computes some values and displays the corresponding results. The real validation process is the union of these two processes. Each process sees only a sub-set of the activities and artifacts: libraries, the debugger and object codes are seen only by the validator; effort drawing, resource allocation reports and history records are visible only to the QA process).

No formalism has currently achieved a consensus, and different Process Engines (PEs) are available, each of which focuses on a given aspect of software process modelling and support.



In the architecture we propose, views are modelled “independently” (see “The Méta-Process.” on page 3) but all views are compiled together to produce the “real process model” in the Internal Formalisms. These internal formalisms are interpreted by different Process Engines which create and maintain different execution contexts for the internal enacting processes. For each defined view, the PSE is in charge of maintaining the corresponding monitoring and interaction interface, based on the view model and the different execution contexts.

2 Process Interface

Our approach is based on the fact that each process fragment has an interface. The concept of interface is borrowed from programming languages. A process interface is the visible part of a process; it defines the process functionalities both in an abstract non executable way and in a formal way; and the consumed and produced artifacts.

The private part contains the implementation. It describes the sub-processes needed, how sub-processes are composed (their ordering.) and coordinated, and how they cooperate.

Collaboration (object synchronization) is supported by the Adele Work Space Manager [5], while cooperation is supported by Work Context [6].

3 The Méta-Process.

Our méta-process follows the following steps.

3.1 Model Building

A model can be built from existing process fragments (their interface). The goal here is essentially to reuse existing fragments.

The concepts of interface and process dependency allows technology developed for Software Configuration Management to be reused: automatic computation of process configuration which ensures the corresponding model will be complete and consistent and that reuse of existing process fragments is maximized.

3.2 Defining Views on a Process.

Once the model has been built, it is worth defining the view(s) needed on this (real) process. At this point in time, the model is indeed a complete model. The agent(s) which will use that process may not need to see the complete process, but only an *executable abstraction* of it i.e. an operational view.

A view is a new process interface built starting from the real process interface, and *hiding* part of the interface. Some parts are *renamed*; other *composed*, i.e. a connex sub part of the process can be abstracted as a single entity, under a single name.

The concept of view, as defined here, has strong links, at least from the product point of view, with the concept of view and virtual object type as defined in recent work on OODB, [7, 8, 9, 10].

3.3 Making Views Operational.

A view is operational if the agent can interact with it as if it were a real process. It means the view must be enactable, and that all aspects, such as monitoring, guidance and performance, must be possible from a view as well as from the complete process.

It means that a bidirectional correspondence must be permanently established between the view and the real process. This correspondence is not trivial, especially when composition of the real process has been performed, and is sometimes impossible (see [9], for a discussion of this topic). If the view is an observatory view, there is no restriction, (the correspondence goes only from the real process to the abstract view); otherwise (actor view) strong restrictions have to be imposed.

3.4 Sharing Process Instances.

In most work, each process fragment, when instantiated, produces a new independent process instance. We think this is an oversimplified view. In fact, most views share process *instances*, in the same way as they share object instances. Obvious examples are meetings, where agents (i.e. processes) share the same process instance (the meeting) or a monitoring view.

In practice it means the same execution context is shared by different processes. Each agent “sees” the same execution context, and just notices some action as being carried out “automatically” when they are performed by another agent of the same process. Sharing processes (and not only the artifacts) is a natural and simple way of implementing cooperative work. Otherwise, explicit cooperation protocols are needed to inform each process of the progresses made by the other, thus introducing unneeded complexity, and worse, the observed process needs to be modified.

We must explicitly define whether activities are shared and private, in almost the same way as we have to define whether the information is shared or private data.

3.5 The Méta-process.

Once the previous issues have been dealt with, the creation of a new process will be undertaken using the following steps:

- 1 Look for the needed processes, querying/browsing the existing interfaces (3.1). If they exist:
 - Select the convenient view of the needed existing process(3.1).
 - If no views are convenient, build the needed view(3.2).
 - Make new views operational (3.3).
 - Define the sharing of process instances (3.4).
- 2 If new process fragments must be defined:
 - Define the new process fragments.
 - Define the convenient view of these fragments (3.2).

4 Current Status

In our Esprit project, PERFECT, we integrated Process Weaver (work flow and petri net based) [6] and Adele (data flow and event based) to create a Virtual Process Engine built from both PEs and a BMS protocol for the communication and coordination of the basic PEs. We then defined a high-level language (APEL which stands for Abstract Process Engine Language) which is compiled towards the corresponding internal formalisms of Adele (triggers) and Process Weaver (tokens and transitions). A part of the architecture presented in section 1 has been tested.

Previous work on the Tempo formalism focused on view-point, but from a product perspective.

We have already implemented, in our PERFECT Esprit project, the way to make different and independent PE collaborate, on a peer-to-peer basis, the way to define a higher-level language compiled toward more basic PEs, and the way to coordinate execution contexts. We have also resolved the view point issue, when limited to the product aspects.

We are now planning to experiments on how to manage views of processes and the sharing of process instances, as well as support for definition and for quality modelling, monitoring and process enhancement views.

5 Conclusion

This approach has the following features.

- Maximum **reuse** of existing models, as seen in step 1.
- Avoidance of **duplication** of process models. In step 2, the process creation involves the checking that no existing processes have a similar description.
- Detection of **inconsistent** description of the same processes. All views are consistent by construction, they only abstract some aspects, they cannot provide inconsistent descriptions.
- Process **sharing** is an explicit feature. It avoids to defining independent activities, with explicit and complex collaboration patterns, when in reality the same process instance is shared.
- **Addition** of or **evolution** of a view, both at model and instance level, does not impact on existing processes.
- **High level formalisms** can be used for view modelling, different lower level formalisms can be used transparently for enaction and execution support.

We believe that the current approaches are lacking with respect to all the current above features, and that this approach could represent significant progress. It is felt that the last three points in particular are fundamental, since experience shows they are the main ways in which processes evolve i.e. by refinement of existing processes (adding a view which adds finer grain sub-processes), and adding services around the core processes (adding control, monitoring, quality, and so on). This approach introduces great flexibility into the global process, since adding views can be done with minimal work (maximum reuse), and minimal impact on other processes (view independence). Much work remains to be done, however.

References

- [1] N. Belkhatir, J. Estublier, and W. Melo. *ADELE-TEMPO: An Environment to Support Process Modelling and Enaction*, volume 3 of *Advanced Software Development*, chapter 8, pages 187–217. John Willey and Son inc, Research Study Press, Tauton Somerset, England, 1994.
- [2] L. J. Osterweil. “Software processes are software too.” In *Proc. of the 9th Int’l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.
- [3] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. “Inconsistency handling in multi perspective specification.” In *Proc. ESEC 93, LNCS 717*, pages 84–99, Garmish, Germany, September 1993.
- [4] M. Verlage. “Multi-view modeling of software processes.” In B. Warboy, editor, *Proc. of European Workshop on Software Process Technology EWSPT3*, volume 635 of *LNCS*, pages 123–127, Villard de Lans, France, February 1994. Springer-Verlag.
- [5] J. Estublier. “The adele work space manager.” Adele Technical Report, available at <ftp:imag.fr>, July 1994.
- [6] C. Fernstrom. “Process Weaver: adding process support to Unix.” In L. Osterweil, editor, *Proc. of the 2nd Int’l Conf. on the Software Process*, pages 12–26, Berlin, Germany, 25 – 26 February 1993. IEEE Computer Society Press.
- [7] E. Rudensteiner. “Multiview: A methodology for supporting multiple views in object oriented databases.” In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
- [8] E. Bertino. “A view mechanism for object oriented databases.” In *Proc. of Int. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.
- [9] A. Geppert, A. Scherrer, and K. Dettrich. “Derived types and subschemas: Toward better support for logical data independence in object oriented data models.” TR 93.27, Institut fur Informatik, Universitat Zurich, 199.
- [10] Q. Chen and M. Shan. “Abstract view object for multiple oodb integration.” In *First JSSST Int. Symposium on Object Technology for Advanced System*, Kanarau, Japan, Nov 1993.

